Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation ▮ (▮▮▮▮) ▮▮▮–▮▮▮
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

Asynchronous iterative sub-structuring methods

Frédéric Magoulès[*], Cédric Venet

CentraleSupélec, Université Paris-Saclay, France

Received 10 October 2014; received in revised form 17 January 2016; accepted 4 May 2016

Abstract

In this paper, a new asynchronous iterative sub-structuring method is presented. This method is based on a classical sub-structuring approach, but during the iterations of the algorithm, at the end of each iteration the synchronisation between the processors is here removed leading to totally asynchronous iterations. The mathematical proof of the convergence of this new asynchronous iterative sub-structuring method is first introduced, followed by an example of the parallel implementation. Numerical results performed on a three dimensional test case illustrate the robustness, performance and efficiency of the asynchronous version over its synchronous counterpart.
© 2016 International Association for Mathematics and Computers in Simulation (IMACS). Published by Elsevier B.V. All rights reserved.

Keywords: Sub-structuring methods; Domain decomposition methods; Iterative methods; Asynchronous iterations; Parallel computing

1. Introduction

The traditional scheme for parallel iterative algorithms is called *synchronous iterations* [38,20]. This describes a method where a new iteration is only started when all the data from the previous iteration has been received. Initially this scheme has been used with *synchronous communication*. This means that data can only be exchanged when the source and destination processes have finished their computation. In most case this leads to a lot of wasted time. An improvement, allowing to overlap the communication time with the computation time, is to use *asynchronous communication*. The only difference is that data can be received by a process while it is still computing the previous iteration. Thus the data computed by a process is sent as soon as the iteration computation is finished. This allows to reduce the communication time a little bit. Further down this path there is *flexible communication*. In this approach the data is sent as soon as possible, i.e., it is sent in parts before the end of the iteration. This is only useful if the data is computed progressively. However, when used it reduces the communication time a lot since the quantity of data to exchange at the end of an iteration is greatly reduced. All these synchronous iterations use the same mathematical model and have the same convergence properties as their sequential counterparts. They have been widely studied and are often simply called parallel iterative algorithms, synchronous being omitted. They are very efficient when used with well balanced workloads and short communication times. However, these conditions are hard and expensive to achieve especially as the number of cores used increases.

[*] Corresponding author.
E-mail address: frederic.magoules@hotmail.com (F. Magoulès).

http://dx.doi.org/10.1016/j.matcom.2016.05.009
0378-4754/© 2016 International Association for Mathematics and Computers in Simulation (IMACS). Published by Elsevier B.V. All rights reserved.

Another kind of iterative algorithm can help to solve these scalability problems. Called first *chaotic relaxations*, it is usually designed by *asynchronous iterations*. The initial work on this scheme was done by Chazan and Miranker [11] and Donnelly [12] and has generated a lot of further research. Among the earliest studies, one can cite the work of Miellou [35,34], Baudet [4], Bertsekas and Tsitsiklis [5,6], El Tarazi [13,14]. There are three commonly used schemes for asynchronous iterations. The first one is called *totally asynchronous iterations*. It sets very few conditions on the iterations and communications except that they must never stop indefinitely. In their book [6], Bertsekas and Tsitsiklis introduce a mathematical model of these as well as several convergence theorems widely applicable. In particular a very general convergence theorem based on a set of imbricated boxes has been used as the basis of numerous convergence results in various domains. This is the scheme which is considered in this paper. The second scheme is called *partially asynchronous iterations*. It is based on the assumption that the communication time is bounded and that each process does at least one iteration every given period. Some useful algorithms which do not converge under totally asynchronous iterations can be proven to converge with these assumptions, see for example Refs. [41,40]. The third scheme is called *flexible asynchronous iterations*. This class of algorithm is more recent [18] and includes in particular the asynchronous two-stage iterative methods [17]. As in the synchronous iterations case, flexible communication means that the data is sent as soon as possible, but due to the strength of asynchronous iteration, it is more general. Data can also be integrated into an iteration as soon as received without waiting for the start of a new iteration. Moreover partial data can also be used, i.e., even if only part of the data from a process (or an approximation of it) has been received, this part can be used immediately without waiting for the rest.

Asynchronous algorithms have been used to solve a variety of numerical problems including the non-exhaustive list: nonlinear fixed points and optimisation problems for partial differential equations and ordinary differential equations [1], flow electrophoresis problems [10], obstacle problems [39], option pricing problems [19,9], and references therein. In this paper, only linear fixed point problems are considered with totally asynchronous iterations. The goal is to solve a linear system of partial differential equations arising from a finite element method. The first asynchronous algorithms used to solve this type of problems are those using contracting operators: Jacobi, Richardson, Successive Over Relaxation method, fixed step gradient descent, etc. More generally, the convergence of fixed point iterations have been studied for arbitrary contracting operators [25,6]. Other asynchronous algorithms for this fixed point problem include the additive Schwarz method [16,21] and more generally the multi-splitting method [7,8] as first introduced by O'Leary and White [36].

The plan of this paper is as follows. Synchronous iterative methods with a particular focus on the Jacobi algorithm, are first introduced in Section 2, followed in Section 3 by their asynchronous counterpart. In Section 4 the classical sub-structuring method is presented, together with a simple parallel implementation. The new asynchronous version of this sub-structuring method is then introduced in Section 5. After the mathematical demonstration of the convergence of this new method, a possible parallel implementation is presented. Numerical results performed on an academic problem, illustrate in Section 6 the robustness, performance and efficiency of this new asynchronous sub-structuring method. Finally, Section 7 concludes this paper.

2. Synchronous iterative methods

2.1. Sequential iterative methods

Iterative methods are based on successive applications of a function on an initial guess of the solution until the result is close enough of the solution [38,20]. The solution $x \in \mathbb{R}^n$ of a linear system $Ax = b$ is here analysed, with $A \in \mathbb{R}^{n\times n}$, a nonsingular $n \times n$ matrix with coefficients in $\mathbb{R}$, and with $b \in \mathbb{R}^n$, the right hand side vector. In the following, the quantity $x_0 \in \mathbb{R}^n$ denotes an arbitrary given value, and x^* denotes the exact solution. To solve this linear system, the matrix A is split into two matrices M and N such as $A = M - N$ where M is a nonsingular matrix. With this splitting, the linear system $Ax = b$ can be rewritten in the form $Mx = Nx + b$ or equivalently $x = M^{-1}Nx + M^{-1}b$ (or similarly $x = Tx + c$, where $T = M^{-1}N$ and $c = M^{-1}b$). The iterative algorithm considered here could be defined by the iterations:

$$Mx(k+1) = Nx(k) + b$$

or equivalently:

$$x(k+1) = Tx(k) + c$$